AJAX

- **What You Should Already Know**
  - Before you continue you should have a basic understanding of the following:
    - HTML / XHTML
    - JavaScript

- **AJAX = Asynchronous JavaScript and XML**

- AJAX is not a new programming language, but a technique for creating better, faster, and more interactive web applications.

AJAX

- With AJAX, your JavaScript can communicate directly with the server, using the JavaScript XMLHttpRequest object.
- With this object, your JavaScript can trade data with a web server, without reloading the page
- AJAX uses asynchronous data transfer (HTTP requests) between the browser and the web server, allowing web pages to request small bits of information from the server instead of whole pages.
- The AJAX technique makes Internet applications smaller, faster and more user-friendly. AJAX is a browser technology independent of web server software.

AJAX is Based on Web Standards

- AJAX is based on the following web standards, and these standards are well defined, and supported by all major browsers. AJAX applications are browser/platform independent.
  - **JavaScript**
  - **XML**
  - **HTML**
  - **CSS**

AJAX is About Better Internet Applications

- Web applications have many benefits over desktop applications;
  - they can reach a larger audience,
  - they are easier to install and support,
  - easier to develop.
- However, Internet applications are not always as "rich" and user-friendly as traditional desktop applications.
- With AJAX, Internet applications can be made richer and more user-friendly.
- There is nothing new to learn.
  - AJAX is based on existing standards.
  - These standards have been used by most developers for several years.

AJAX Uses HTTP Requests

- In traditional JavaScript coding, if you want to get any information from a database or a file on the server, or send user information to a server, you will have to make an HTML form and GET or POST data to the server.
- The user will have to click the "Submit" button to send/get the information, wait for the server to respond, then a new page will load with the results.
- Because the server returns a new page each time the user submits input, traditional web applications can run slowly and tend to be less user-friendly.
- With AJAX, your JavaScript communicates directly with the server, through the JavaScript **XMLHttpRequest** object With an HTTP request, a web page can make a request to, and get a response from a web server - without reloading the page.
- The user will stay on the same page, and he or she will not notice that scripts request pages, or send data to a server in the background.

The XMLHttpRequest Object

- By using the XMLHttpRequest object, a web developer can update a page with data from the server after the page has loaded!
- AJAX was made popular in 2005 by Google (with Google Suggest).
- Google Suggest is using the XMLHttpRequest object to create a very dynamic web interface:
  - When you start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions.
- The XMLHttpRequest object is supported in Internet Explorer 5.0+, Safari 1.2, Mozilla 1.0 / Firefox, Opera 8+, and Netscape 7.

- **AJAX - Browser Support**
  - The keystone of AJAX is the XMLHttpRequest object.

```
<html>
<body>
<script type="text/javascript">
function ajaxFunction()
{
    var xmlHttp;  //Different browsers use different methods to create the XMLHttpRequest object

    try     // Firefox, Opera 8.0+, Safari
    {
        xmlHttp=new XMLHttpRequest();
    }
    catch (e)
    {
        try  // Internet Explorer
        {
            xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e)
        {
            try
            {
                xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch (e)
            {
                alert("Your browser does not support AJAX!");
                return false;
            }
        }
    }

    xmlHttp.onreadystatechange=function()
    {
        if(xmlHttp.readyState==4)
        {
            document.myForm.time.value=xmlHttp.responseText;
        }
    }

    xmlHttp.open("GET","time.php",true);
    xmlHttp.send();
}
</script>

<form name="myForm">
Name: <input type="text" onkeyup="ajaxFunction();" name="username" />
Time: <input type="text" name="time" />
</form>
</body>
</html>
```

Did you notice something missing from the HTML form?

- There is no submit button.

AJAX - More About the XMLHttpRequest Object

- Before sending data to the server, we have to explain three important properties of the XMLHttpRequest object.

1. The onreadystatechange Property

- After a request to the server, we need a function that can receive the data that is returned by the server.
- The onreadystatechange property stores your function that will process the response from a server.
- This is not a method, the function is stored in the property to be called automatically.
- The following code sets the onreadystatechange property and stores an empty function inside it:

```
xmlHttp.onreadystatechange=function()
{
   // We are going to write some code here
}
```

2. The readyState Property

- When a request to a server is sent, we want to perform some actions based on the response.
- The onreadystatechange event is triggered every time the readyState changes.
- The readyState property holds the status of the XMLHttpRequest.
- Three important properties of the XMLHttpRequest object:

Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 404: Page not found

- In the onreadystatechange event, we specify what will happen when the server response is ready to be processed.
- When readyState is 4 and status is 200, the response is ready:

```
xmlhttp.onreadystatechange=function()
{
 if (xmlhttp.readyState==4 && xmlhttp.status==200)
   {
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
   }
}
```

Using a Callback Function

- A callback function is a function passed as a parameter to another function.
- If you have more than one AJAX task on your website, you should create ONE standard function for creating the XMLHttpRequest object, and call this for each AJAX task.
- The function call should contain the URL and what to do on onreadystatechange (which is probably different for each call):

```
function myFunction()
{
    loadXMLDoc("ajax_info.php",function() {
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
        }
    });
}
```

3. The responseText Property

The data sent back from the server can be retrieved with the responseText property.

In our code, we will set the value of our "time" input field equal to responseText:

```
xmlHttp.onreadystatechange=function()
{
  if(xmlHttp.readyState==4)
  {
     document.myForm.time.value=xmlHttp.responseText;
  }
}
```

AJAX - Sending a Request to the Server

The XMLHttpRequest object is used to exchange data with a server.

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

Method	Description
open(*method,url,async*)	Specifies the type of request, the URL, and if the request should be handled asynchronously or not. *method*: the type of request: GET or POST *url*: the location of the file on the server *async*: true (asynchronous) or false (synchronous)
send(*string*)	Sends the request off to the server. *string*: Only used for POST requests

GET or POST?

- GET is simpler and faster than POST, and can be used in most cases.

- However, always use POST requests when:
  - A cached file is not an option (update a file or database on the server)
  - Sending a large amount of data to the server (POST has no size limitations)
  - Sending user input (which can contain unknown characters), POST is more robust and secure than GET

GET Requests

A simple GET request:

Example

```
xmlhttp.open("GET","demo_get.php",true);
xmlhttp.send();
```

In the example above, you may get a cached result.

To avoid this, add a unique ID to the URL:

Example

```
xmlhttp.open("GET","demo_get.php?t=" + Math.random(),true);
xmlhttp.send();
```

If you want to send information with the GET method, add the information to the URL:

Example

```
xmlhttp.open("GET","demo_get2.php?fname=Henry&lname=Ford",true);
xmlhttp.send();
```

POST Requests

A simple POST request:

Example

```
xmlhttp.open("POST","demo_post.php",true);
xmlhttp.send();
```

To POST data like an HTML form, add an HTTP header with setRequestHeader(). Specify the data you want to send in the send() method:

Example

```
xmlhttp.open("POST","ajax_test.php",true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xmlhttp.send("fname=Henry&lname=Ford");
```

Method	Description
setRequestHeader(*header*,*value*)	Adds HTTP headers to the request. *header*: specifies the header name *value*: specifies the header value

The url - A File On a Server

- The url parameter of the open() method, is an address to a file on a server:

```
xmlhttp.open("GET","ajax_test.php",true);
```

- The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

Asynchronous - True or False?

- AJAX stands for Asynchronous JavaScript and XML, and for the XMLHttpRequest object to behave as AJAX, the async parameter of the open() method has to be set to true:

```
xmlhttp.open("GET","ajax_test.php",true);
```

- Sending asynchronous requests is a huge improvement for web developers.
- Many of the tasks performed on the server are very time consuming.
- Before AJAX, this operation could cause the application to hang or stop.

- With AJAX, the JavaScript does not have to wait for the server response, but can instead:
  - execute other scripts while waiting for server response
  - deal with the response when the response ready

Async=true

- When using async=true, specify a function to execute when the response is ready in the onreadystatechange event:

- **Example**

```
xmlhttp.onreadystatechange=function()
 {
 if (xmlhttp.readyState==4 && xmlhttp.status==200)
  {
  document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
  }
 }
xmlhttp.open("GET","ajax_info.php",true);
xmlhttp.send();
```

Async=false

- To use async=false, change the third parameter in the open() method to false:

  ```
  xmlhttp.open("GET","ajax_info.php",false);
  ```

- Using async=false is not recommended, but for a few small requests this can be ok.
- Remember that the JavaScript will NOT continue to execute, until the server response is ready.
- If the server is busy or slow, the application will hang or stop.
- Note: When you use async=false, do NOT write an onreadystatechange function - just put the code after the send() statement:
- **Example**

```
xmlhttp.open("GET","ajax_info.txt",false);
xmlhttp.send();
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

Server Response

To get the response from a server, use the responseText or responseXML property of the XMLHttpRequest object.

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

The responseText Property

If the response from the server is not XML, use the responseText property.

The responseText property returns the response as a string, and you can use it accordingly:

Example

```
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

The responseXML Property

If the response from the server is XML, and you want to parse it as an XML object, use the responseXML property:

Example

Request the file cd_catalog.xml and parse the response:

```
xmlDoc=xmlhttp.responseXML;
txt="";
x=xmlDoc.getElementsByTagName("ARTIST");
for (i=0;i<x.length;i++)
  {
  txt=txt + x[i].childNodes[0].nodeValue + "<br>";
  }
document.getElementById("myDiv").innerHTML=txt;
```